DII.3006.NT40.PG-1

Defense Information Infrastructure (DII)

Common Operating Environment (COE)

Version 3.0.0.6

Programming Guide (Windows NT 4.0)

February 5, 1997

Prepared for:

Defense Information Systems Agency

Prepared by:

Inter-National Research Institute (INRI)
12200 Sunrise Valley Drive, Suite 300
Reston, Virginia 20191

Table of Contents

Preface 1

1. Writing Programs Using the COE Tools 3
1.1 Overview 3
1.2 Additional Sources of Information 4

2. Application Development Overview 5
2.1 Writing Your Application with the DII COE APIs 5
2.2 Building Your Application with the DII COE APIs 5
2.3 Running Your Application 5

3. Segment Development 7
3.1 Segment Layouts 7
3.2 COE Tools Overview 8
3.2.1 Running the COE Tools From the Command Line 8
3.2.2 COE Runtime Tools 8
3.2.3 COE Developer Tools 8
3.3 Building Your Segment 9
3.3.1 Identifying and Creating Required Subdirectories 9
3.3.2 Creating and Modifying Required Segment Descriptor Files 10
3.3.3 Installing a Segment 12
3.4 Customizing Your Segment 13
3.4.1 Adding Menu Items 13
3.4.2 Adding Icons 19
3.4.3 Reserving a Socket 20
3.4.4 Displaying a Message 21

Appendix A - Sample Segments 23
A.1 Sample Account Group Segment Layout 23
A.2 Sample Software Segment Layout 25

Appendix B - Verifying Segment Syntax and Loading a Segment onto a Floppy Diskette 27
B.1 Running VerifySeg Against the Sample Segment 28
B.2 Running TestInstall Against the Sample Segment 28
B.3 Running TestRemove Against the Sample Segment 29
B.4 Running MakeInstall Against the Sample Segment 29

Appendix C - Installing the Developer's Toolkit 31

List of Tables

Table 1. Segment Descriptor Files 10
Table 2. SegInfo Descriptor Sections 11

List of Figures

Figure 1. Segment Directory Structure 7

Preface

The following conventions have been used in this document:

[HELVETICA FONT]	Used to indicate keys to be pressed. For example, press [RETURN].
`Courier Font`	Used to indicate entries to be typed at the keyboard, Windows NT commands, titles of windows and dialog boxes, file and directory names, and screen text. For example, execute the following command: `A:\setup.exe`
"Quotation Marks"	Used to indicate prompts and messages that appear on the screen.
Italics	Used for emphasis.

This page intentionally left blank.

1. Writing Programs Using the COE Tools

1.1 Overview

This document provides an introduction to the capabilities of the Defense Information Infrastructure (DII) Common Operating Environment (COE) tools for the DII COE Version 3.0.0.6 for Windows NT Operating System Version 4.0. These tools consist of a set of runtime tools and a set of developer's tools.

This document has been designed to help developers start using the DII COE tools. This document explains the basic use of the tools, regardless of whether they are run from a menu or from the command line.

The document consists of the following sections and appendices:

Section/Appendix	Page
Application Development Overview Provides an overview of how to write, build, and run an application.	5
Segment Development Discusses the different types of segments and the process of segment creation.	7
Sample Segments Describes how to install sample segments, which can be used to test segment installation and execution.	23
Verifying Segment Syntax and Loading a Segment onto a Floppy Diskette Provides examples of how to convert a segment to the *DII COE Integration and Runtime Specification* segment format, verify segment syntax, temporarily install and remove a segment, and load a segment onto an installation floppy diskette.	27
Installing the Developer's Toolkit Describes how to load the Developer's Toolkit, which contains the components needed to create segments that use DII COE components.	31

Descriptions assume familiarity with the C programming language and with the Windows NT development environment.

1.2 Additional Sources of Information

Reference the following documents for more information about the DII COE toolkit:

- *Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification* Version 2.0, DII COE I&RTS:Rev 2.0, Inter-National Research Institute, October 23, 1995

- *Defense Information Infrastructure (DII) Common Operating Environment (COE) Version 3.0.0.6 Application Programmer Interface (API) Reference Guide (Windows NT 4.0),* DII.3006.NT40.RG-1, Inter-National Research Institute, February 5, 1997.

2. Application Development Overview

Developers may require access to public Application Programmer Interfaces (APIs) in order to ensure an application complies with the *DII COE Integration and Runtime Specification*. To use these public APIs, developers must (1) include the public `include` files with the DII COE tools header and (2) compile and link the application with the Developer's Toolkit `include` directory and libraries. Public APIs are documented in the *DII COE API Reference Guide (Windows NT 4.0)*.

2.1 Writing Your Application with the DII COE APIs

To access the DII COE tools through the provided APIs, you must include the following header in your application:

```
#include <DIITools.h>
```

The standard location for the Developer's Toolkit header is:

```
DII_DEV\include
```

2.2 Building Your Application with the DII COE APIs

To build your application with the DII COE APIs, you must link your application with the `COECom.lib`, `COESeg.lib`, `COETools.lib`, and `COEUserPrompts.lib` libraries, which are on the DII COE Developer's Toolkit floppy diskettes.

The standard location for the Developer's Toolkit libraries is:

```
DII_DEV\libs
```

To compile the application, make sure the `include` file path in the compile environment includes `<local path>\DII_DEV\include`. Also make sure the library (`lib`) path includes `<local path>\DII_DEV\libs`.

2.3 Running Your Application

The DII COE provides the foundation and infrastructure in which one or more applications run. Applications must be formatted properly as segments to operate under the COE. The segment is the basic building block of the COE runtime environment. A segment is a collection of one or more Computer Software Configuration Items (CSCIs) that are managed most conveniently as a unit. Segments generally are defined to keep related CSCIs together so functionality easily may be included or excluded. All applications must be put in the DII COE runtime environment segment format to be installed onto a DII COE-compliant machine.

Once an application has been put in the proper segment format, the segment can be installed in a disciplined way through instructions contained in files provided with each segment. These files are called segment descriptor files and are contained in a special subdirectory, `SegDescrip`, which is called the segment descriptor subdirectory. Installation tools process the segment descriptor files to create a carefully controlled approach to adding or deleting segments to or from the system.

Once installed, your application can be invoked in the DII COE environment in two ways: (1) running your application from a command shell window or (2) invoking your application from an icon. The easiest way to test your application is to invoke it in a command shell window. This gives you easy access to your application for debugging purposes and allows you to check any diagnostic information your application is generating. Section 3.4, *Customizing Your Segment*, describes how to set up your application to be invoked as a menu item or as an icon.

3. Segment Development

The following subsection discusses the different types of segments and the process of segment creation. Reference Section 5.0, *Runtime Environment*, of the *DII COE Integration and Runtime Specification* for a more detailed explanation of segments.

3.1 Segment Layouts

In the DII COE approach, each segment is assigned a unique, self-contained subdirectory. DII COE compliance mandates specific subdirectories and files underneath a segment directory. These subdirectories and files are shown in Figure 1. Six segment types exist: Account Group, COTS (Commercial Off-the-Shelf), Data, Database, Software, and Patch. The precise subdirectories and files required depend on the segment type. Some of the subdirectories shown in Figure 1 are required only for segment submission and are not delivered to an operational site.

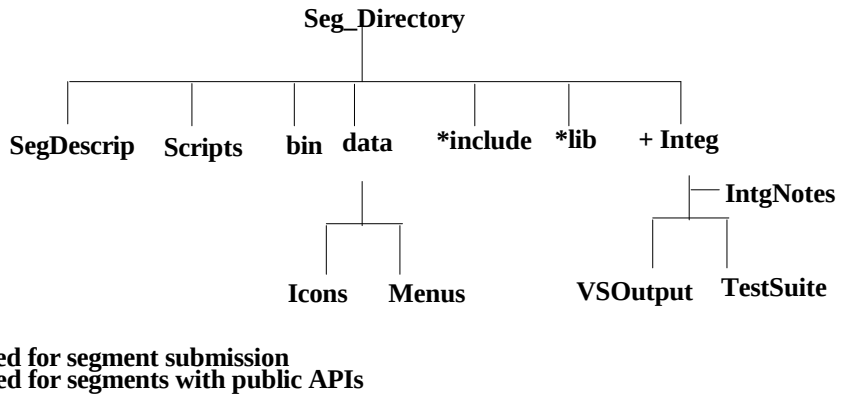


Figure 1. Segment Directory Structure

The following runtime subdirectories are normally required, depending on the segment type: (1) `SegDescrip`, which is the directory containing segment descriptor files; (2) `bin`, which is the directory containing executable programs for the segment; and (3) `data`, which is the subdirectory containing static data items, such as menu items, that are unique to the segment, but that will be the same for all users on all workstations.

The `SegDescrip` directory is required for every segment because it contains the installation instructions for the segment. A segment cannot modify files or resources outside its assigned directory. Files outside a segment's directory are called community files. COE tools coordinate modification of all community files at installation time. Reference Section 5.5, *Segment Descriptors*, of the *DII COE Integration and Runtime Specification* for a detailed explanation of `SegDescrip` files.

3.2 COE Tools Overview

The COE tools were constructed to aid developers in the creation and ultimate installation of DII COE segments. All tools can be run from the command line.

3.2.1 Running the COE Tools From the Command Line

This section provides a brief overview of running the COE tools from the command line. Reference the following sources for detailed information about the COE tools: Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification*; the Developer's Toolkit release notes; the on-line help files for the tools; and the help page provided by the tools.

When run from the command line, the tools are designed to run interactively and accept one or more command line parameters.

The tools are used to communicate with the outside world in two ways. First, the tools use the exit function to set the DOS status variable. The status return value is set to `0` for normal tool completion or to `255` if an error occurs. A status return value greater than `0` but less than `255` indicates a completion code that is tool specific. The `ERRORLEVEL` function may be used to examine a tool's exit status.

Second, the tools use `stdin` and `stdout` and thus support input and output redirection. Redirecting `stdin` allows the tools to receive input from a file or from another program, while redirecting `stdout` allows the tools to provide output to other programs.

NOTE: Redirecting `stdin` is not always convenient. The `-R` command line parameter allows a tool to read input from a response file instead of from `stdin`.

For example, the following statement can be used to write the results of VerifySeg to a file:

```
VerifySeg -p d:\testsegs Seg1 > results.txt
```

3.2.2 COE Runtime Tools

Reference Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification* for a complete description of the DII COE runtime tools.

3.2.3 COE Developer Tools

Windows NT tools are delivered on a floppy diskette and may be copied to any directory desired for development. The TestInstall and TestRemove tools must be run as `Administrator` because they modify files the user may not own. VerifySeg should also be run as `Administrator`, although it is not mandatory. The VerifySeg tool requires the user to have write permission to the segment against which the tool was executed.

Reference Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification* for a complete description of DII COE developer's tools.

3.3 Building Your Segment

A segment must be built in a disciplined way using instructions contained in files provided with each segment. These files are contained in a special directory, `SegDescrip`, which is the segment descriptor subdirectory.

This section describes a process to turn an application into a segment so it can be a part of the DII COE. As described earlier, a segment is a collection of one or more CSCIs most conveniently managed as a unit.

3.3.1 Identifying and Creating Required Subdirectories

There are six segment types: Account Group, COTS, Data, Database, Software, and Patch. The following subdirectories normally are required:

Subdirectory	Description
`SegDescrip`	Subdirectory containing segment descriptor files. This directory is always required for every segment and contains the installation instructions for the segment. A segment is not allowed to directly modify any files for resources it does not *own*; in other words, a segment cannot modify files or resources outside an assigned directory. The DII COE tools coordinate the modification of all community files at installation time.
`bin`	Executable programs for the segment. These files can be the result of a compiled program or as a result of shell scripts, depending on the type of the segment.
`data`	Subdirectory for static data items, such as menu items, that are unique to the segment, but that will be the same for all users on all workstations.

Reference Sections 5.0-5.5 of the *DII COE Integration and Runtime Specification* for a detailed explanation of segment directory layout and a description of each `SegDescrip` file.

3.3.2 Creating and Modifying Required Segment Descriptor Files

Segment descriptor files are the key to providing seamless and coordinated systems integration across all segments. Reference Table 1 to determine which descriptor files are required for each segment type. For example, the AcctGrp segment requires `ReleaseNotes`, `SegInfo`, `SegName`, and `VERSION` descriptor files in the `SegDescrip` directory, while the Patch segment requires the `PostInstall` descriptor file in addition to the previously listed files. Some segment descriptor information is provided in the files listed in Table 1.

NOTE: In Table 1, `Aggregate` and `COE Comp` are segment attributes that can be associated with any type of segment.

File	Acct Grp	Aggregate	COE Comp	COTS	Data	DB	S/W Seg	Patch
DEINSTALL	O	O	O	O	O	O	O	O
FileAttribs	O	O	O	O	O	O	O	O
Installed	I	I	I	I	I	I	I	I
PostInstall	O	O	O	O	O	O	O	R
PreInstall	O	O	O	O	O	O	O	O
PreMakeInst	O	O	O	O	O	O	O	O
ReleaseNotes	R	R	R	R	R	R	R	R
SegChecksum	I	I	I	I	I	I	I	I
SegInfo	R	R	R	R	R	R	R	R
SegName	R	R	R	R	R	R	R	R
Validated	I	I	I	I	I	I	I	I
VERSION	R	R	R	R	R	R	R	R
R - Required Software		O - Optional		I - Created by Integrator or Installation				

Table 1. Segment Descriptor Files

Other segment descriptor information is arranged within subsections of the `SegInfo` file. As with the descriptor files themselves, some subsections of the `SegInfo` file are required and others are optional depending on the type of segment. Table 2 defines required and optional sections for each segment type.

Section	Acct Grp	Aggregate	COE Comp	COTS	Data	DB	S/W	Patch
AcctGroup	R	O	N	N	N	N	N	N
COEServices	O	O	O	O	O	O	O	O
Community	O	O	O	O	O	O	O	O
Comm.deinstall	O	O	O	O	O	O	O	O
Compat	O	O	O	O	O	O	O	N
Conflicts	O	O	O	O	O	O	O	O
Data	N	N	N	N	R	N	N	N
Database	X	X	X	X	X	X	X	X
Direct	O	O	O	O	O	O	O	O
FilesList	O	O	O	R	O	O	O	O
Hardware	R	R	R	R	R	R	R	R
Icons	R	O	N	O	N	N	O	O
Menus	R	O	N	O	N	N	O	O
ModName	*	*	*	*	*	*	*	*
ModVerify	*	*	*	*	*	*	*	*
Network	N	N	N	N	N	N	N	N
Permissions	O	O	N	N	N	N	O	O
Processes	O	O	O	O	N	N	O	O
ReqrdScripts	N	N	N	N	N	N	N	N
Requires	O	O	O	O	O	O	O	O
Security	R	R	R	R	R	R	R	R
SegType	*	*	*	*	*	*	*	*
R - Required O - Optional N - Not Applicable X - Reserved for Future * - Obsolete								

Table 2. SegInfo Descriptor Sections

3.3.3 Installing a Segment

Follow the procedures below to install a segment after it has been created.

Run VerifySeg

The VerifySeg tool must be run during the development phase to ensure segments use segment descriptor files properly. Run the VerifySeg tool whenever a segment is created or modified. When VerifySeg is run to verify a segment, a `Validated` file is created. Reference Appendix C, *COE Tools,* of the *DII COE Integration and Runtime Specification* for further information about using VerifySeg.

Run TestInstall

Executing the TestInstall tool is not a mandatory step in the installation process, but it is recommended. TestInstall simulates an installation of a segment on the developer's workstation before actual installation. Reference Appendix C, *COE Tools,* of the *DII COE Integration and Runtime Specification* for further information about using TestInstall.

Run TestRemove

Executing the TestRemove tool is not a mandatory step in the installation process, but it is recommended. TestRemove simulates a deinstallation of a segment on the developer's workstation after TestInstall has been run. Reference Appendix C, *COE Tools,* of the *DII COE Integration and Runtime Specification* for further information about using TestRemove.

Run MakeInstall

The MakeInstall tool is used to write one or more segments to an installation media and to package the segment(s) for distribution. MakeInstall checks if VerifySeg has been run successfully on each of the segments and aborts with an error if it has not. Reference Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification* for further information about using MakeInstall.

Run COEInstaller

The COEInstaller tool installs a segment from floppy diskette. Reference Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification* for further information about using the COEInstaller.

3.4 Customizing Your Segment

Most properly designed segments will not require any extensions to the COE, although the segments may need to add menu items and icons. Some segments may need to add special extensions. The following subsections describe how to add menu items, icons, and special extensions.

3.4.1 Adding Menu Items

Menu files are maintained by the DII COE, but no DII COE applications read menu files in this release. Nevertheless, user programs may use their own menu files through this feature.

Menu Entry Format

The Menu Descriptor in the `SegInfo` file is used to specify the name of the segment's menu file and the name of the affected segment's menu file.

The menu bar, pull-down menus, and cascade menus, as well as the menu items they contain, are built according to the menu description entries. The format of the entries is in ASCII with colon-separated fields. Colons are used as delimiters, and spaces are allowed in the fields. Each line ends in a colon with no extra data. A `#` symbol in the first column of a line denotes a comment line. Comment entries may be placed anywhere in the entry and are not processed by the parser.

Valid keywords are `PDMENU`, `PDMENUEND`, `ITEM`, `PRMENU`, `CASCADE`, `CASCADEEND`, `APPEND`, `APPENDEND`, and `SEPARATOR`. Since the Menu Description Entry is based on your menu design, you might not use of all of these keywords. For example, if your menu does not have separator lines, your Menu Description Entry will not contain a `SEPARATOR` keyword.

Each keyword is described in the following paragraphs.

A **`PDMENU`** line contains the following elements:

```
PDMENU: name : enable flag : id # :
```

`PDMENU`	Keyword that indicates the start of a pull-down menu.
`name`	Text used to name the menu. The menu name is displayed on the menu bar.
`enable flag`	Integer value that indicates if a menu is enabled or disabled. The enable flag is `1` if a menu is enabled or `0` if it is disabled. A disabled menu means that no options under that pull-down menu can be selected.

`id#` Optional integer value that provides a unique ID number for the menu. The `PDMENU id#` value must be unique within the menu description file. An absolute value may be provided. However, the `id#` field should be left empty so that relative numbering is used by default.

With relative numbering, an `id#` of `R1` (or leaving the field blank) sets the menu's ID number to `1` plus the `id#` of the last menu processed. An `id#` of `R2` sets the menu's ID number to `2` plus the `id#` of the last menu processed.

The following is an example of a `PDMENU` line:

```
PDMENU: Map Options : 1 : R1 :
```

A **`PDMENUEND`** line contains the following element:

```
PDMENUEND:
```

`PDMENUEND` Optional keyword that indicates the end of a group of pull-down menu items. If `PDMENUEND` is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than `ITEM` or `PRMENU`) is encountered.

The following is an example of a `PDMENUEND` line:

```
PDMENUEND:
```

An **`ITEM`** line contains the following elements:

```
ITEM: name : command : execution type : enable flag : # instances : id# :
check value : security char : autolog flag : print flag : disk flag :
```

`ITEM` Keyword that indicates a menu item description line.

`name` Text used to name the menu item. The item name is displayed in the pull-down menu.

`command` Program with space-separated arguments that is launched if the menu item type is a program. Otherwise, the menu item is called as an application callback. Because callback functions must be linked into the same executable as the menu bar, applications cannot use callbacks when adding items to the system menu bar.

execution type — Integer value that indicates how to execute a command, as follows:

- 1 = executable program
- 2 = void callback function with no parameters (not yet implemented)
- 3 = Motif callback function (not yet implemented).

enable flag — Integer value that indicates if a menu item is enabled or disabled. The enable flag is 1 if a menu item is enabled or 0 if it is disabled. A disabled menu item means that the option cannot be selected.

instances — Integer value used to set the maximum number of times the item can be executed simultaneously.

id# — Optional integer value that provides a unique ID number for the menu item. Each ITEM id# entry must be unique within a PDMENU listing. (ITEM entries in a PRMENU must be unique within that PRMENU.) Reference the id# description under the PDMENU keyword listing.

check value — Optional integer value that sets the star and checks annotations of a menu item. Possible values are:

- 0 = no annotation (default)
- 1 = visible check mark
- 2 = check mark, but not visible
- 3 = visible star
- 4 = star member, but not visible.

security char — Optional character value used to determine the lowest security level under which a menu item can be classified. Valid settings are:

- N = No Classification
- U = Unclassified (default)
- C = Confidential
- S = Secret
- T = Top Secret.

autolog flag — Optional character value, T or F, used to indicate if the command should be logged automatically.

print flag — Optional character value, T or F, used to indicate if the command should have a print capability.

disk flag — Optional character value, T or F, used to indicate if the command should have a disk access capability.

NOTE: The following elements are not yet fully implemented: `check value`, `autolog flag`, `print flag`, and `disk flag`.

The following is an example of an `ITEM` line:

```
ITEM: Netscape : Netscape.. : 1 : 1 : 1 : R1 : 0 : T : F : F : F :
```

A **`PRMENU`** line contains the following elements:

```
PRMENU: name : enable flag : id# :
```

`PRMENU`	Keyword that indicates a cascading menu button. It is used to mark where a cascade menu is to be connected to an upper-level menu.
`name`	Text used to name the cascade menu with which to connect. The `PRMENU` name is displayed in the pull-down menu.
`enable flag`	Integer value that indicates if a cascade menu is enabled or disabled. The enable flag is `1` if a cascade menu is enabled or `0` if it is disabled. A disabled cascade menu means that menu options on the cascade menu cannot be selected.
`id#`	Optional integer value that provides a unique ID number for the cascading menu. Each `PRMENU id#` must be unique within a `PDMENU` listing. Reference the `id#` entry under the `PDMENU` keyword listing.

The following is an example of a `PRMENU` line:

```
PRMENU: Software : 1 : R1 :
```

A **`CASCADE`** line contains the following element:

```
CASCADE: name :
```

`CASCADE`	Keyword that indicates the start of a cascade menu. The cascade menu connects to the `PRMENU` entry of the same name.
`name`	Text used to name a cascade menu. The name is used to attach a cascade menu to a cascading menu button. This name must be the same as the name field in the `PRMENU` entry.

The following is an example of a CASCADE line:

```
CASCADE: Software :
```

A **CASCADEEND** line contains the following element:

```
CASCADEEND:
```

CASCADEEND	Optional keyword that indicates the end of a group of cascade menu items. If CASCADEEND is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than ITEM or PRMENU) is encountered.

The following is an example of a CASCADEEND line:

```
CASCADEEND:
```

An **APPEND** line contains the following elements:

```
APPEND: name :
```

APPEND	Keyword that indicates the start of a group of items to append to an existing menu. The menu will be created if it does not already exist. The group is appended to the PDMENU or CASCADE entry of the same name.
name	Text used to select the menu to which a group of items is appended.

The following is an example of an APPEND line:

```
APPEND: Options :
```

An **APPENDEND** line contains the following element:

```
APPENDEND:
```

APPENDEND	Optional keyword that indicates the end of a group of menu items to be appended to an existing menu. If APPENDEND is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than ITEM or PRMENU) is encountered.

The following is an example of an APPENDEND line:

```
APPENDEND:
```

A **SEPARATOR** line contains the following element:

```
SEPARATOR:
```

SEPARATOR	Optional keyword that indicates that a Motif separator widget is to be placed in a menu at the point where the keyword occurs.

The following is an example of a SEPARATOR line:

```
SEPARATOR:
```

Example of Adding a Menu Item

To add menu items, include the Menus Descriptor in the SegInfo Segment Descriptor file. Specify the Menu file you use wish to load and the Menu file you wish to update. The Menu file you wish to load should be located in the TstSeg\data\Menu directory, assuming the segment name is TstSeg. This example will add the Test Program menu item to the Software menu under the SysAdm account group.

The program TSTCOEAskUser_example will be executed when invoked through the menu item:

SegInfo (menu descriptor only)

```
[Menus]
TstSegMenu:SA_Default.main
TstSegMenu
#--------------------
# Software Menu Items
#--------------------
APPEND          :Software
ITEM            :Test Program   :TSTCOEAskUser_example:1:1:1:R1
APPENDEND :
```

Also include the $SEGMENT keyword in the SegName Segment Descriptor file to specify the name of the affected segment. In this case it is System Administration.

SegName

```
#
# SegName For Test Segment
#
$TYPE:SOFTWARE
$NAME:Test Segment
$PREFIX:TST
$SEGMENT:System Administration:SA:/h/AcctGrps/SysAdm
```

3.4.2 Adding Icons

Icon Entry Format

The Icon Description Entry contains information on all icon-based processes. The entry, or set of entries, to be used is passed to the Program Manager.

Icons are built using the icon section in the `SegInfo` file. The entry is a specially formatted icon description that has colon-separated fields. The colons are used as delimiters, and spaces are allowed in the fields. Each line ends in a colon with no extra data. A # symbol in the first column of a line denotes a comment line. Comment entries may be placed anywhere in the file and are not processed by the parser.

The format of the icon entry is as follows:

```
ICON file : affected icon file
```

The affected icon file contains information about both the icon and the executable. The format of the file is as follows:

```
Icon Name : Icon File : Executable File : Comments
```

Where `Icon Name` is the title placed next to the icon in the `Programs` menu, `Icon File` is the icon image file, `Executable File` is the executable to be launched by the Program Manager, and `Comments` is the comment line. The `Icon File` field is optional. If an `Icon File` is specified, the file must be located in the segment's `data\Icons` directory. The `Executable File` must be located in the segment's `bin` directory.

An example of an affected icon file is as follows:

```
Edit Profiles : EditProf.ico: EditProf.exe
```

Icons are added to a `Programs` menu group named for the account group to which they belong.

Example of Adding an Icon

To add an icon, include the `Icons` Descriptor in the `SegInfo` Segment Descriptor file. Specify the icon file you wish to load. This file should be located under the `TstSeg\data\Icons` directory, assuming the segment directory is `TstSeg`. This example will add the `Test Program` icon to the SysAdm account group. The program `TSTCOEAskUser_example` will be executed when invoked through the icon.

SegInfo (icon descriptor only)

```
[Icons]
TstSegIcons:SA_Default
```

TstSegIcons

```
#--------------------
# Software Icons
#--------------------
Test Program :TestProgramIcon:TSTCOEAskUser_example
```

Also include the `$SEGMENT` keyword in the `SegName` Segment Descriptor file to specify the name of the affected segment. In this case it is `System Administration`.

SegName

```
#
# SegName For Test Segment
#
$TYPE:SOFTWARE
$NAME:Test Segment
$PREFIX:TST
$SEGMENT:System Administration:SA:/h/AcctGrps/SysAdm
```

3.4.3 Reserving a Socket

To add a service, include the `COEServices` Descriptor in the `SegInfo` Segment Descriptor file. Also include the `$SERVICES` keyword in the `SegInfo` Segment Descriptor file to specify the service to be added. If the port number requested is already in use under another name, an error will be generated.

NOTE: Port numbers in the range 2000-2999 are reserved for DII COE segments.

SegInfo (COEServices descriptor only)

```
[COEServices]
#
#  This is my service to add
#
$SERVICES
irc_ser:3001:upd
```

3.4.4 Displaying a Message

This subsection shows an example of how to display a message during the PostInstall process. Five runtime tools can be used to communicate with the user: COEAskUser, COEInstError, COEMsg, COEPrompt, and COEPromptPasswd. These tools may be used to display information to the user or to ask the user a question and, based on the result, perform different actions.

In this example, the user is asked questions using the COEAskUser runtime tool, which is described in Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification*.

```
rem ======================================================
rem PostInstall script for Segment Tst
rem ======================================================

echo "Performing PostInstall"

COEAskUser -B "RED LAN" "BLUE LAN" "Which LAN will you be connecting to?"

if ERRORLEVEL 1 goto RED

if ERRORLEVEL 0 goto BLUE

goto END:

:RED
echo "Connecting to RED LAN"
rem
rem Perform some action based user input
rem
goto END:

:BLUE
echo "Connecting to BLUE LAN"
rem
rem Perform some action based on user input
rem

:END
echo "Done with PostInstall"
```

This page intentionally left blank.

Appendix A - Sample Segments

This appendix includes a segment layout for a sample Account Group segment (GCCS) and a sample Software segment (Seg1). These basic templates of typical DII COE segments can be used to test segment installation and execution.

NOTE: If you do not have the GCCS sample Account Group segment installed on your machine, you will receive several warnings indicating that it must be installed before the Seg1 sample Software segment can be loaded.

The Seg1 sample Software segment will add the `Seg1 Hello World` icon to the GCCS Account Group in the `Programs` menu. The program `Seg1_HelloWorld.exe` will be executed when invoked through the icon.

Reference Appendix B, *Verifying Segment Syntax and Loading a Segment onto a Floppy Diskette,* for instructions on how to validate the Seg1 sample Software segment and load the segment onto a floppy diskette.

A.1 Sample Account Group Segment Layout

The layout of the GCCS sample Account Group segment is:

```
gccs

   SegDescrip
      DEINSTALL.BAT
      PostInstall.bat
      ReleaseNotes
      SegInfo
      SegName
      VERSION
```

The `SegDescrip` files contain the following:

DEINSTALL.BAT

```
REM ======================================================
REM
REM DEINSTALL
REM
REM Routine to perform necessary actions when segment
REM is deinstalled.
REM
REM ======================================================
```

PostInstall.bat

```
REM ======================================================
REM
REM PostInstall
REM
REM Routine to perform necessary actions after segment
REM has been loaded.
REM
REM ======================================================
```

ReleaseNotes

```
This is a sample Account Group.
Other Segments will need to extend the environment
as they add their specific functionality
to the account group.
```

SegInfo

```
#============================================================
#
#   Account Group SegInfo file.
#
#============================================================

[AcctGroup]
GCCS Operator:350::1:GCCS:GCCS Default
$CLASSIF:UNCLASS

[Hardware]
$CPU:PC
$OPSYS:NT
$DISK:500
$MEMORY:100

[Security]
UNCLASS
```

SegName

```
#============================================================
#
#   Account Group SegName file.
#
#============================================================
$TYPE:ACCOUNT GROUP
$NAME:GCCS COE
$PREFIX:GCCS
```

VERSION

```
3.0.0.6:2/5/97
```

A.2 Sample Software Segment Layout

The layout of the Seg1 sample Software segment is:

```
Seg1
   bin
      Seg1_HelloWorld.exe
   data
      Icons
         TestIcons
   SegDescrip
      DEINSTALL.BAT
      PostInstall.BAT
      ReleaseNotes
      SegInfo
      SegName
      VERSION
```

The `SegDescrip` files contain the following:

DEINSTALL

```
REM ======================================================
REM
REM DEINSTALL
REM
REM Routine to perform necessary actions when Seg1
REM is deinstalled.
REM
REM ======================================================
```

PostInstall.BAT

```
REM ======================================================
REM
REM PostInstall
REM
REM Routine to perform necessary actions after Seg1 Test
REM Segment has been loaded.
REM
REM ======================================================
```

ReleaseNotes

```
This is the Seg1 Test Segment.
```

```
SegInfo
#============================================================
#
#   DII Database Admin Segment SegInfo
#   Descriptor file.
#
#============================================================
[Hardware]
$CPU:PC
$OPSYS:NT
$DISK:500
$MEMORY:100

[Icons]
TestIcons

[Security]
UNCLASS

SegName
#============================================================
#
#   DII Seg1 Test Segment
#   Descriptor file.
#
#============================================================
$TYPE:SOFTWARE
$NAME:Test Segment #1
$PREFIX:Seg1
$SEGMENT:GCCS COE:GCCS:/h/AcctGrps/GCCS

VERSION
3.0.0.6:10/15/96
```

The `data\Icons` file contains the following:

```
TestSegIcons
TstSegIcons
#--------------------
# Software Icons
#--------------------
Test Program :TestProgramIcon.ico:TSTCOEAskUser_example
```

Appendix B - Verifying Segment Syntax and Loading a Segment onto a Floppy Diskette

This appendix provides examples of how to verify segment syntax, install a segment temporarily, and load a segment onto an installation floppy diskette. The segment verification and loading process involves the following steps:

STEP 1: **Run the VerifySeg tool**. Run VerifySeg to validate that the segment conforms to the rules for defining a segment (i.e., to verify the segment syntax).

STEP 2: **Run the TestInstall tool**. Run TestInstall against the sample segment to install the segment temporarily. This step is optional; if you choose not to run TestInstall, proceed to STEP 4.

STEP 3: **Run the TestRemove tool**. Run TestRemove against the sample segment to remove the segment temporarily installed in STEP 2.

STEP 4: **Run the MakeInstall tool**. Run MakeInstall to load the segment onto floppy diskette. After the segment is loaded onto floppy diskette, it is ready to be installed using the `DII Installer` icon from the System Administration group.

Subsections B.1-B.4 show how to perform these steps against the Seg1 sample Software segment, which is described in Appendix A, *Sample Segments*.

NOTE: In the following subsections, the VerifySeg, TestInstall, TestRemove, and MakeInstall tools are being run against the Seg1 sample Software segment. The output of each command will vary depending on the segment being converted. Note the following severity indicators:

`(F)` indicates a FATAL ERROR	`(D)` indicates a DEBUG statement
`(W)` indicates a WARNING	`(V)` indicates a VERBOSE statement
`(E)` indicates an ERROR	`(O)` indicates an ECHO statement.

NOTE: In the following subsections, boldface text indicates information that the user must input.

B.1 Running VerifySeg Against the Sample Segment

```
*************************************************************************
VerifySeg -p \SampleSegs Seg1

Results of verification (/SampleSegs/Seg1) :
   Totals
   ------
   Errors:    0
   Warnings:  0
*************************************************************************
```

B.2 Running TestInstall Against the Sample Segment

```
*************************************************************************
TestInstall -p \SampleSegs Seg1
****************************************
TestInstall - Version 1.0.0.7
****************************************
The following options have been selected:
****************************************
Print warning messages.
******************************
Segments to be TestInstalled:
******************************
Segment: seg1 Path: P:\SampleSegs
*************WARNING*************
TestInstall may modify COE files already in use.
This may cause unpredictable results if COE processes are already running.
Make sure no other COE processes are running before using TestInstall.
Do you want to continue with the TestInstall? (y/n):y
Processing seg1

  Successfully ran preprocessor on segment seg1
No PreInstall script for segment seg1
Do you want to run PostInstall for Segment seg1? (y/n):y
REM ======================================================
REM
REM PostInstall
REM
REM Routine to perform necessary actions after Seg1 Test
REM Segment has been loaded.
REM
REM ======================================================
Successful Installation of seg1
*************************************************************************
```

B.3 Running TestRemove Against the Sample Segment

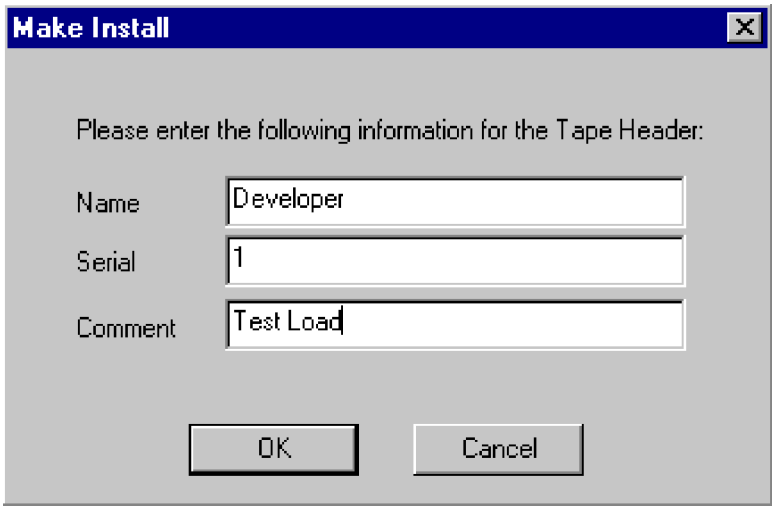
```
*******************************************************************************
TestRemove -p \SampleSegs Seg1
*************WARNING*************
TestRemove may modify COE files already in use.
This may cause unpredictable results if COE processes are already running.
Make sure there are no other COE processes running before using TestRemove.
Do you want to continue with the TestRemove? (y/n):y
SETTING P:\SampleSegs\Seg1 FOR INSTALL_DIR

REM ======================================================
REM
REM  DEINSTALL
REM
REM  Routine to perform necessary actions when Seg1
REM  is deinstalled.
REM
REM ======================================================
*******************************************************************************
Successful Removal of Seg1
```

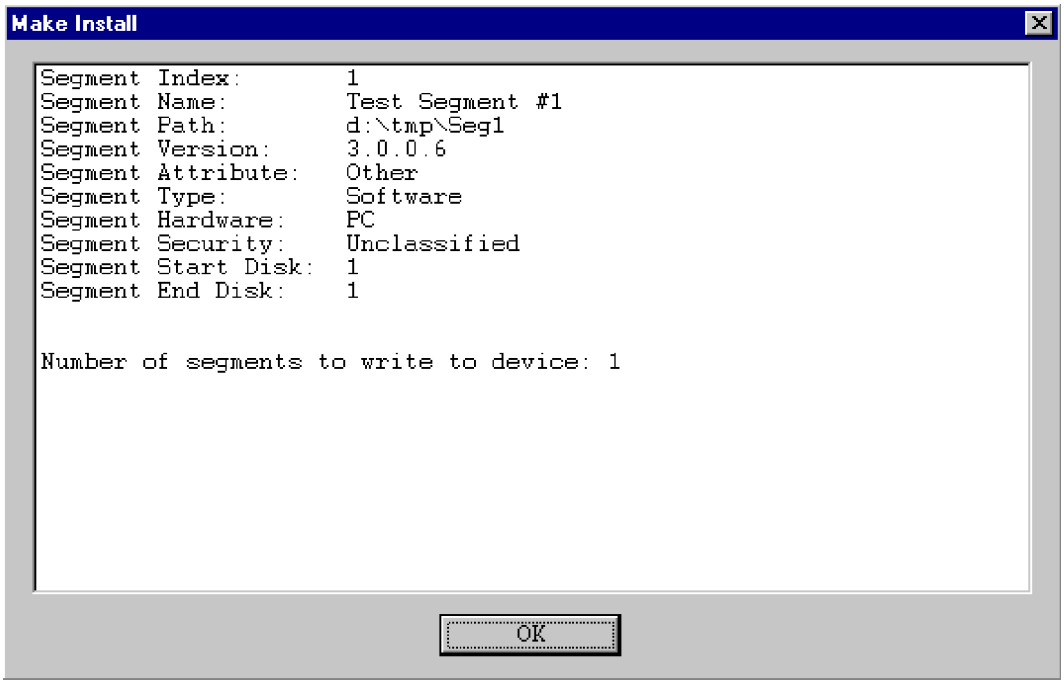
B.4 Running MakeInstall Against the Sample Segment

The example below shows the three windows displayed by MakeInstall as it loads segments onto a floppy diskette. The MakeInstall tool does not generate any command window output. The Seg1 sample Software segment is used in the example. These windows should appear in the order shown below.

```
*******************************************************************************
MakeInstall -p d:\tmp Seg1
```




Make Install
To Continue Please Select OK.
WARNING: Existing data may be overwritten
OK
Cancel


Make Install
Segment Index: 1
Segment Name: Test Segment #1
Segment Path: d:\tmp\Seg1
Segment Version: 3.0.0.6
Segment Attribute: Other
Segment Type: Software
Segment Hardware: PC
Segment Security: Unclassified
Segment Start Disk: 1
Segment End Disk: 1
Number of segments to write to device: 1
OK

Appendix C - Installing the Developer's Toolkit

The Developer's Toolkit contains the components needed to create segments that use DII COE components. Developer's tools are delivered on floppy diskettes separate from the DII COE Kernel. These tools can be run from the command line.

Follow the steps below to install and run the DII COE 3.0.0.6 Developer's Toolkit for Windows NT 4.0:

STEP 1: **Choose a directory for installation**. Choose a directory on your hard drive into which the tools can be installed.

NOTE: These developer's tools are not location sensitive and can be moved to any directory desired for development; however, the provided Dynamic Link Libraries (DLLs) must remain in the same directory with the tools that require them or must be in a location that is also included in the `PATH` environment variable. Therefore, it is recommended that the tools remain in the same directory (i.e., `DII_DEV\bin`).

STEP 2: **Copy the `DII_DEV` directory to the desired location**. Use the Windows NT Explorer to copy the contents of the `DII_DEV` directory from the Developers' Toolkit floppy diskettes to the desired location.

NOTE: To copy a directory, click on the `Start` button on the task bar, point to `Programs`, and click on the `Windows NT Explorer` option. The `Exploring` window appears. Click on the directory you want to copy to highlight it and then select the `Copy` option from the `Edit` pull-down menu. Open the destination directory and select `Paste` from the `Edit` pull-down menu. The directory, then, is copied to the new location.

STEP 3: **Add an entry in your `PATH` environment variable**. Add an entry in your `PATH` environment variable for the `DII_DEV\bin` directory. To add this entry, click on the `Start` button on the task bar, point to `Settings`, and click on `Control Panel`. The `Control Panel` window appears. Double-click on the `System` icon to open the `System Properties` window. Click on the `Environment` tab.

If the `PATH` environment variable appears in the `User Variables for Administrator` panel, highlight it. The word `Path` should appear in the `Variable` field and a list of one or more paths separated by semicolons should appear in the `Value` field. Place your cursor at the end of this list of paths and type `;[directory]\DII_DEV\bin` in the `Value` field, where `[directory]` is the directory on your hard drive in which you installed the developer's toolkit.

If the `PATH` environment variable does not appear in the `User Variables for Administrator` panel, type `Path` in the `Variable` field and type `[directory]\DII_DEV\bin` in the `Value` field.

Click on the `Set` button to establish the path as a setting in the `User Variables for Administrator` panel. Then click on the `OK` button to exit the window.

STEP 4: **Create or copy a DII COE segment onto your hard drive**. Reference the *DII COE Integration and Runtime Specification* for information about creating a segment in the DII COE format.

STEP 5: **Run VerifySeg**. Type the following command to run VerifySeg:

```
VerifySeg -p [segment path] [segment directory]
```

STEP 6: **Run TestInstall**. Type the following command to run TestInstall:

```
TestInstall -p [segment path] [segment directory]
```

STEP 7: **Run TestRemove**. Type the following command to run TestRemove:

```
TestRemove -p [segment path] [segment directory]
```

STEP 8: **Run MakeInstall**. Type the following command to run MakeInstall:

```
MakeInstall -p [segment path] [segment directory]
```